STPC CLIENT

BIOS WRITER'S GUIDE

Revision 1.1

STMicroelectronics
Technoparc du Pays de Gex -B.P. 112
165, rue Edouard Branly
01637 Saint Genis Pouilly (France)

Revised January 23, 1999

STMicroelectronics values your feedback. Please send it and any recommendations you may have regarding this document to:

STPC.support@st.com **with the subject line "BWG11".**

Author: M. SANCHES

Table of Contents

TECHNICAL SUPPORT

STMicroelectronics is on the Internet with a World Wide Web (WWW) site on which product presentation, technical literature as well as product support information can be found.

A dedicated STPC section is available providing up to date hardware documentation and software tools.

The Web and e-mail addresses are:

WWW: *http://www.st.com/stpc*

e-mail: *stpc.support@st.com*

This page is left intentionally blank

i What's in the package ?

The following documents are provided with this guide:

stpc.def	misc and general defines and equate for STPC.
bwcode.asm	486 assembly sample code
bwdata.asm	x86 assembly for data used by the sample code.
inta1.inc	Int a1 sample code.
inta1.def	Inta1 related equates.
pirq.exe	Dos program to test pci irq mapping
testa1.exe	Dos program to test the int a1.
dclk.exe	Dos program to test the gclk programing (dclk 255).

ii How to use these document and sample source code ?

Within this BIOS Writer's Guide document you will not find any more sources, only references to sample code assembly proc, data, table and index within table. All those references are marked as described below:

proc:
TBD (InitChip_SP)
data table:
TBD (TabInitIDe)

Index within a table:
TBD (AnyBootInitTable index 0x21)

Within the sample code file look for ';PORT' or ';PORTING' assembly comment to find where porting is required. These may be only note regarding specific implementation, code to be replaced or modified with bios specific call or just because code is missing such as cmos data retrieving.

iii Important note for previous guide revision users

The @Read_22_23 and @write_22_23 macros have changed if you have written any code based on them it will not work. A fast solution is to changed all the new @Read_22_23 and @Write_22_23 by @ReadStpc and @WriteStpc and keep the old macros unchanged.

iiii What's new from 1.0 to 1.x ?

- The sample code is adapted to match new macros.
- The Memory auto sizing code is modified to support 128Meg the frame buffer size is not any more programmed there.
- Separated dram refresh setup from the InitChip.
- Init table minors changes.
- CPU cache write back setup added.
- Isa memory hole setup sample code added.
- Board specific GPIO features added, Enable disable flash sample code.

1. Memory

1.1. DRAM Refresh

Setting up the DRAM refresh is the first task to be complete prior to use any dram.
The refresh period have been calculated according to DRAM specification for each CPU clock freq and result in the DRAM_Refresh table of bgdat100.asm file. The InitFresh_SP stackless proc in bgwcode.asm can be called directly to do all the job.

See DRAM_Refresh, InitRefresh_SP

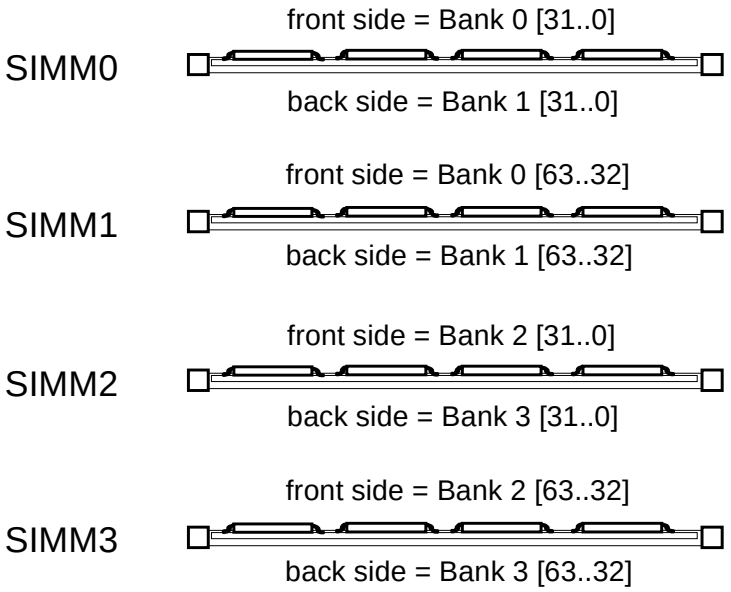
1.2. Memory Autosizing

- Memory Sizing

Prerequisite : DRAM Refresh on with the appropriate frequency, 4GB access limit, ROM execution.
Goals : Setup the memory controller to drive the full memory on board, with the correct size and type.

- About memory Bank & size

The STPC Client can handle both 32 bits or 64 bits memory Bank width. Organize as for SIMM slot. The SIMM slots can be either single-sided or double-sided, and are mapped to the memory as described below:



Each Bank top address and Bank width starting by Bank 0 must be initialized with the total amount of memory in megabyte -1 and set to the correct width.

- Example 1

if SIMM0/SIMM1 are 8Mbyte single-sided
it will result in Bank0 64-bit wide with Bank0 top address = 2*8 -1 = 0xF
since Bank 1/2/3 are empty Bank 1, Bank 2 & Bank 3 top address will be 0xF (Bank empty)

- Example 2

if SIMM0/SIMM1/SIMM2/SIMM3 are 8Mbyte single-sided
it will result in Bank0/Bank2 64-bit wide with
Bank0 top address = 2*8 - 1 = 0xF
Bank1 top address = 2*8 - 1 = 0xF (Bank0 top + 0 Bank 1 empty)
Bank2 top address = 0xF + 2*8 = 0x15(Bank2 = 2*8)
Bank3 top address = 0x15 + 0 = 0x15 (Bank2 top + 0 Bank3 empty)

- Example 3

if SIMM0/SIMM1 are 8Mbyte double-sided
it will result in Bank0/Bank1 64bit wide with
Bank0 top address = 2*4 -1 = 0x7 (8Meg /2 by side=4)
Bank1 top address = 0x7+2*4= 0xF (bk0 top + 8)
Bank2 top address = 0xF+0 = 0xF
Bank3 top address = 0xF+0 = 0xF

- How to automatically find Bank size (top address) and width?

We must start with Bank0, and assume first that all the Banks are 64-bit wide (set appropriate bit in the memory Bank width register). Since we do not know the size of Bank0 nor the other, we set them all to the maximum size: 128M-1 = 0x7F. To find if the Bank is really 64-bit wide writing at an Address and Address+4 must succeed. If Bank isn't 64-bit wide, the write access at address+4 will fail. Then the top address is found by checking when the memory wrap happens. The value is setup to the Bank and we continue with next Bank up to the last one.

The algorithm described above, is implemented by AutoSizeDram_SP. Since dram mapping will change while performing the autosizing the code cannot run from ram but only from ROM, it mean being sure shadow is off before to execute it. Usually on soft reset there is no need to resize dram (even if frame buffer size have changed in cmos).
Note also that after executing these proc the frame buffer (VGA memory) is 0. So, at that moment, it is strongly recommended here to program it at its final value.

See *AutoSizeDram_SP, Init4GBSeg_SP, GDT_BASE*

- Memory Type/Timings

The STPC memory controller allows timing programming for:

RAS precharge time
RAS to CAS delay
CAS Pulse width

It provides also programming of memory type (FPM/EDO) as well as Bank width (32/64-bits) for each single Bank.
Finally the controller can be programmed to keep RAS active or not (all Bank only), this feature allows memory request in the same page not requiring a RAS reassertion so saving the RAS precharge time on each new accesses.

Since those timings refer to CPU clock, there is one setting for each CPU clock value authorized.

1.3. DRAM Timing

We have computed and tested DRAM timings for all the possible CPU clock, you can find them organized as table in bgdat100.asm (FPM60nsTimingTable,FPM70nsTimingTable, EDO60nsTimingTable, EDO70nsTiming).
We also provided SetMemTimings_DI stackless proc to perform dram type and speed programing, these proc accept bank number, dram type and speed as input parameters.

See FPM60nsTimingTable, FPM70nsTimingTable, EDO60nsTimingTable, EDO70nsTimingTable, DramBkTimingRegIndex, label,SetMemTimings_DI

At power on the typical boot DRAM init sequence will go like that:

```
            @call_sp     InitChip_SP
            @call_sp     InitRefresh_SP
            @call_sp     Init4GBSeg_SP
            @call_sp     AutosizeDram_SP
;program the frame buffer size now
            ;specific load of ah with frame buffer size from cmos
            ; in case cmos is invalid the default value 4 (512K) should be used
            mov          al, FB_SIZE_IDX
            @write_22_23
;perform some more init if required.....
            ;specific dl load with dram type speed from cmos
            xor          bh, bh       ;reset bank
NextBank:
            @call_Di     SetMemTimings_DI
            inc          bh
            cmp          bh, 3
            jbe          NextBank
```

2. CPU Interface

2.1. Getting local BUS speed

The STPC On Chip CPU frequency is selected using external jumpers setting. This information can be read trough strap option register 0x5F bit[5..3]. The @GetBusSpeed macro in stpc.def file is returning these information and can be used wherever CPU clock frequencies dependencies are required (DRAM/ timing, refresh).

See @GetBusSpeed macro and HCLK_FREQ_XX define in stpc.def

2.2. Cacheable memory

The highest cacheable memory address is given by the size of the L2 cache, so when L2 is disabled (in all the STPC product) this size must be setup to the highest size to allow all the DRAM to be cacheable so this mean to set reg 0x21 bit[7..5] to 101b.
Note that this is done by InitChip_SP when using the default InitTable.

See AnyBootInitTable reg index 021h

2.3. Write back cache

The STPC CPU core and chipset support Write back mode for L1 cache with full snoop support from ISA or PCI agent. To make this feature work and anyway before to use the L1 cache in WB mode several register need programing. First at CPU level in CCR2 (reg index C2) Bit 1 must be set (CPU snoop pins enabled), bit 5 should be cleared (no data write back when hold asserted). NW bit in cr0 register must be set also to enable WB mode otherwise WT is used. Then at chipset level in Chip register index 0x20 bit 5 must be set (CPU support L1 WB).

Note: all the programing regarding chipset/cpu WB is done by the IniChip_SP proc

For compatibility issue when OS when disable cache they clear CD but also NW at the same time, the problem is that they never turn NW on again. These make the CPU finally run in WT mode far less efficient than WB. So it may be better with some OS to lock NW once programmed. This is done by EnableL1WB sample code in bwgcode.asm.

See EnableL1WB, AnyBootInitTable

2.4. Settings for best performances

Enabling the CPU NoLock feature allow more data to become cacheable and so improve system performances the feature is disabled by default in IniChip_SP. These is done by setting CCR1 (index 0xC1) bit 4.
Typically:

```
mov         al, 0C1h
call        Read_22_23
or          ah, 10h
call        Write_22_23
```

Until not enabled the CPU does not performed any burst write cycle, to enable these features bit 6 of CCR2 (index 0xC2) must be set. These feature is enable by default in InitChip_SP.

See AnyBootInitTable reg index 0C2h and 0C1h

The Host (CPU) to memory speed is based on the L2 cache controller setting, register 0x20 and 0x22. For up to 75Mhz the following values have to be used to achieve best performances:

Table 1:

Register	Value
Ox20	Ox28
Ox21	OxA0
Ox22	Ox81

3. Chipset

3.1. Shadow Memory

- E000,C000,D000 segments

The shadow memory is handled as 16 KByte separate block for segments C,D and E. The Shadow read/write attribute can be programmed separately, so it's possible to directly shadow an area without using a RAM buffer by directly reading and writing to the same address.

Warning in the init table, segments C, D and E shadow are disabled by setting the registers (offset) 0x25 0x26 and 0x27 to 0.

In the sample code we provide the EnableShadow16k proc for enabling one of the 16k bloc in the wanted mode. These code use the STPCShadowTable of BWGDATA.ASM witch might be directly usable in many bios where shadow is table driven.

See STPCShadowTable, ShadowStruct, EnableShadow16k, AnyBootInitTable
reg index 25h,26h,27h

- F000 segment

The BIOS area shadow is handled as a unique 64K area and can be enabled for reading or writing separately.
This area isn't part of the table because in case of soft reset the code should already run in shadow. Disabling it may result in a crash since runtime code is most of the time totally different from the ROM image (non shadowed code) that is executed only once at hard reset. If it is necessary to run from ROM, the shadow should be turned off, then the code should do a jump as a cache flush to ensure that code is reloaded from ROM.

```
;this code assume to be at the same place in rom and shadow ram
          mov          ax, 0028h              ; disable shadow F000
          @Write_22_23
          jmp          FlushPipe              ; make sure now fetched from ROM
                                              ; not shadow after soft reset
align 16
FlushPipe:
```

- Shadow memory cacheable attribute

The 32K at C000 as the 64k System BIOS at F000 can be cached by setting the bits 5 and 6 in register 0x28.

3.2. Safe boot init table

The AnyBootInitTable in BWGDATA.ASM hold most of the chipset register that should be reprogrammed or set to their default values at reset or soft reset time. Depending on BIOS implementation some of the registers (shadow, ISA clk...) should be handled separately.
These table is used by InitChip_SP proc witch should be called as soon as possible during post. What it perform is just programming the Init table.

See AnyBootInitTable, InitChip_SP

3.3. Getting PCI clock

- Determining the PCI clock value

With STPC Client chip it is possible to retrieve the host (CPU) clock as the PCI divider ratio, so it is really simple to compute the PCI clock.
The GetPciClock proc provided as sample use the CPU speed and PCI clk divider not to know the exact PCI clock but getting a value so it can be used to as an index to access data in a table that defines PCI clock.

See HclkPll2FreqTable, GetPciClock

3.4. Gate A20, Fast Reset

The STPC can emulate the A20 gating as well as the CPU Fast Reset generally implemented in the keyboard controller. These two lines used to be driven by the KBDC, read or write accesses being performed by the standard KBD 0xD1 (write) or 0xD0 (read) commands. CPU Fast Reset can also be performed by writing 0xFE directly in port 0x64.
When emulation is on, the STPC traps any 0xD1 write to port 0x64 and puts the following 0x60 port write in the A20 gate and CPU reset lines. Command 0xFE is also trapped and performs a CPU reset.
Note that both accesses below are not forwarded to the KBDC.

Since command 0xD0 is not trapped or handled by the STPC there is no way to know the state of these 2 lines. Also as the commands are not forwarded to the KBDC there is no need to wait for it to be ready to program either the Gate A20 or the reset.
Note that, because there is no way to bring KBDC A20 mask and reset lines into the STPC, the keyboard emulation must always be on (should be turned off temporarily) or if it is not the CPU A20 line will always be masked.

Sample code for fast Gate A20' and CPU fast reset is provided in BWGCODE.ASM.

See EnableA20Fast, DisableA20Fast, CpuFastReset

4. IDE

4.1. Enable/Disable OnBoard IDE

To disable the STPC IDE it might not be enough to disable the PCI IDE device. Both primary and secondary unit must to be set in native mode first (bit 0 and 1 of PCI conf reg offset 9 set). To separately enable the primary and secondary channel it's just enough to enable the PCI device and put the desired controller in Legacy mode, these is assuming the other channel to be in native mode with valid PCI IO address space programed in the base address registers (offset 0x10 to 0x23).
The DisableOnChipIde sample code disable the onchip IDE, but take care that these proc will reprogrammed the base address, in case it is not wanted just remove the concerned pci configuration index from the TabInitIde (0x10 to 0x23). For a PnP aware bios if want to keep the programmed value the following address space from 0xFFD8 to 0xFFFF is used for the IDE and must be reserved.

For enabling the primary controller and respectively the secondary the EnableOnChipIde1 and EnableOnChipIde2d can be used.

On a typical BIOS the DisableOnChipIde proc will first be called, then if no Primary external IDE is detected the EnableOnchipIde1 will be called and then same for secondary channel.

See DisableOnChipIde, TabInitIDE, EnableOncHipIdeIde1,EnableOnChipIde2

Note: The Onchip primary and secondary IDE interrupt are hardwired to IRQ 13 and 14.

4.2. PIO/Busmaster timings

The OnChip PCI IDE automatically generated timings for the read and write access to the HDD, Those timings are fully programmable and need so to be correctly programed to match the PIO specs and achieve the best HDD throughput.

See SipIdeRecoveryTablePtr, SipIdeRecoveryTable_33, SipIdeRecoveryTable_30, SipIdeRecoveryTable_25, SipIdeRecoveryTable_20, ProgSipIdePio, PioRecoveryTranslationTable, PioActiveTranslationTable

5. PCI

5.1. PCI address space

The only valid address for PCI device is located from 0x10000000 up to 0x1FFFFFFF (128M to 256M).
All of the unused system memory address range can also be used for PCI, for example if there is 32MB on the board the 32M to 128M locations might be used to locate any PCI device.
Also when Onboard VGA is enabled the following area from 0x10000000 0x101FFFFF (16M starting at 128M) should be reserved as graphics Frame buffer. This memory is, by default, not mapped or used by the VGA, it is only enabled by STPC OS Graphics drivers, so it may or may not be reserved depending on final use.

5.2. Linking PCI INT request to the Interrupt Controller

To map a PCI INT to the PIC (Programmable Interrupt Controller 8059), the interrupt line number has to be written in the appropriate register:

Reg 0x52 for PCI INTA
Reg 0x53 for PCI INTB
Reg 0x54 for PCI INTC
Reg 0x55 for PCI INTD

Bit 7 of each of these registers must be set to 1 to enable the link.

Note that not all the IRQ values are valid for mapping PCI INT. The reserved ones are 0,1,2,8,0xd and should not be used.

Also PCI interrupt lines are not edge sensitive like on the ISA bus but level active. This results in reprogramming of the associated interrupt line in register 0x56 or 0x57.
The PciRoutInt sample code manages all the settings. The PCI INT# and PIC line are required as well as the Enable status as input argument.

See `PciRoutInt`

5.3. PCI slot/Device number/ PCI INT line relations

Internally the STPC implements Device Number 0xB, 0xC. Device number 0xB is assigned to the Host to PCI bridge.

Note: the related Host to PCI bridge class code is wrong.

Device number 0xC is assigned to a multiple-function device where:

- Function 0: PCI to ISA bridge
- Function 1: Bus master IDE controller

The 3 OnBoard PCI connectors (in our reference schematics as in all our evaluation boards) are assigned device numbers 0x1D, 0x1E and 0x1F. The table below describes how each slot PCI INT is connected to the STPC PCI INT line input:

Table 2:

	SLOT INT				
Device No	**INT A**	**INT B**	**INT C**	**INT D**	**Slot**
0x1F	**INT A**	**INT B**	**INT C**	**INT D**	**1**
0x1E	**INT B**	**INT C**	**INT D**	**INT A**	**2**
0x1D	**INT C**	**INT D**	**INT A**	**INT B**	**3**

As an example we have in Slot 2 a board that requires an interrupt. For example this could be INT A. Consequently, the board is device No 0x1E and it's INT A is connected to STPC INT B (see table). Finally we should route PCI INT B to the desired interrupt line.

For testing the correct handling of PCI we provide pirq.exe dos program that display PCI to IRQ mapping and sensitivity programmed into the chip.

6. ISA

6.1. Bus speed

The STPC ISA bus clock can be set either to a fixed value 7.16MHz (14.318MHz/2) or to the PCI clock divided by 4, this is programmed in register 0x50 bit 4 where '0' mean 14MHz/2 and '1' PCI clk/4.

If the ISA is to be set to 14MHz/2 the ISA synchronizer must be enabled because the ISA and PCI clock are not synchronous. This is done by programming reg 0x59 bit 0 and it must be done before enabling the 14MHz/2 in reg 0x50.

For PCI clk /4 the ISA synchronizer can be disabled to have better performances and the order in which it is done is not relevant.

The SetIsaClk sample code programmed both registers to set the ISA clock to the value passed as input parameters.

See SetIsaClk, AnyBootInitTable reg index 50h and 59h

Note: PCI clock/4 can result in a high clock frequency value (>8MHz), this is the case if PCI clock is greater than 33MHz which should not happen but can occur using CPU clock >= 75MHz and a PCI divider set to 2 (PCI clock= 37.5MHz or more).

6.2. IPC Specific timings

The following parameters regarding the IPC (DMA controller, Interrupt controller,... also known as 82C206) can be programmed separately in register 0x01:

- Wait state for IPC register access (1,2,3 or 4)
- Wait state in 16 bits DMA transfer (1,2,3 or 4)
- Wait state in 8 bits DMA transfer (1,2,3 or 4).
- IO read and MEM write cycle assertion behavior (at the same clock or one clock after).
- DMA clock either ISA or ISA / 2 To be PC compatible this should be set to ISA clock / 2.

The 0x0 value has to be used to get the best performances.

One extra wait state for ISA cycle can also be added/removed by programming bit 5 of register 0x50.
For best performances this bit should be set to 0 (no extra wait state).

6.3. EPROM Access and ROM Chip Select

The STPC built-in logic provides a Chip select feature to the external Onboard EPROM that is highly configurable. Whenever the CPU accesses the ROM from C000:0 up to F0000:FFFF the STPC can provide the chip select avoiding additional logic for extension BIOS or software placed in the System BIOS EPROM/FLASH.

The Chip select generation for F000 segment is obviously always active and cannot be turned off. But Chip select assertion for access to another segment is software controlled and can be turned on or off as many times as wanted. Each of the 64K rom area from C000:0 up to E0000:FFFF are controlled separately in register 0x51 where:

Bit 0 Control Chip select assertion for C000 segment
Bit 1 Control Chip select assertion for D000 segment
Bit 2 Control Chip select assertion for E000 segment

Let us take an example: Using a 256K EPROM that integrates a 64K extension BIOS 0xD000 and a System BIOS code at E000. To make those 2 segments accessible, register 0x51 should be Or-masked with 110b. The reset value can be set in AnyBootInitTable reg index 51h.
Notice that when the shadow read attribute of those areas are enabled read accesses are performed from shadow memory and not from the ROM whatever the value of the corresponding ROM sharing.

See AnyBootInitTable reg index 0x51

6.4. ROM Writing protection (flash)

The STPC also provides ROM write access protection. When the feature is enabled any write accesses to the EPROM (F000 as the other enabled segments) are not passed to the ISA bus, so for Onboard EPROM programming these protections must be turned off by setting register 0x51 bit 3 to 0.
In the same way as for reading, if the shadow write attribute is set for an area the write accesses will go to the shadow memory prior to the ROM, and this bit will not be prevent write accesses to the shadowed ROM memory.

Note: When no write accesses to the FLASH are required, this bit must be set to "1". In particular, this setting has to be kept during normal operation. That’s why it is enabled in the Init table.

Typically bios have Flash enable and disable standard proc, we provide EnableFlash_DI and DisableFlash_DI as sample which disable/enable the rom protection and enable/disable the 12V VPP to the program the Flash.

See EnableFlash_DI, DisableFlash_DI, AnyBootInitTable index 50h, 7bh

6.5. ISA memory Hole

With STPC it is possible to open a memory address space (named hole) that fall into the isa bus. This hole can be from 1 MegaByte up to 8 Mega byte and locate anywhere from 1 Meg to 15Meg. The only constrain is the base address that must be aligned to the size of the hole.

For enabling or disabling the feature as to programmed the base and size we provide the SetIsaHole proc. It accept as parameter the size (0 mean disabled) and the base address. This proc verify the validly of the input parameters and will return with an error status if something isn't correct.

Note: The isa hole is disable by InitChip_SP.

See SetIsaHole, AnyBootInitTable reg index 24h

7. On-Chip VGA

7.1. Enable/Disable Onchip VGA

For some reason the system BIOS might have to access OnChip VGA or video registers. As an Add-in VGA board might be present, the On-chip VGA should be either turned off or on. This is the purpose of the two functions EnableOnChipVGA and DisableOnChipVGA provided as sample:

See EnableOnChipVga, DisableOnChipVga

- On-Chip Video Disable

The STPC integrates video processing and VGA overlay features that should be disabled on soft reset by the bios. Since the video registers are made accessible through VGA registers the On-Chip VGA must be first turned on. As those registers which are memory mapped somewhere around the 128M, so the 'flat model' (4GB segment limit in real mode) should be turned on to be able to access them. Once done the VGA should be turned off again.

It should proceed as follows:

call EnableOnChipVga
call DisableVideo
call DisableOnChipVga

If, for some reason, stack or flat model is not available, the code in those functions can be copied and executed in sequence to avoid call.

- Palette Snoop

The purpose is not to discuss here about PCI VGA palette snoop which is part of standard PCI features but about Onchip VGA Palette snoop.

When Onchip VGA is used palette write cycle can be forwarded (register 0x29 bit 2 set) or not (register 0x29 bit 2 cleared) to the PCI bus.

```
mov     al, 29h
@Read_22_23
or      ah, 100b        ;  enable palette snoop
; and   ah,NOT 100b     ;  disable palette snoop
@Write_22_23
```

Note: Register 0x29 bit 2 is blindly cleared by the Enable/Disable OnChip VGA code

- VGA Interrupt Routing

The STPC VGA and video interrupt (also named VMI) is routed in the same way as the PCI interrupt in register index 0x58. It is not a requirement for the BIOS to route the On-Chip VGA interrupt especially when the OnChip VGA is disabled. It will only free IRQ out of the PCI PnP. So we recommend not to route it in any case, OS driver should do the job if it is required.

7.2. Graphic clock

The STPC OnChip VGA as other Onchip video processing is based on the UMA architecture where CPU and video have there own DRAM controller operating with different clocks.

For the video the base clock is GCLK which is provided by an On-Chip programmable PLL that must be correctly setup.

The faster the clock runs, the better the performance which can be achieved but these values are not the same for FPM and EDO memories. Also it is useful to be able to force a very slow and safe clock as a backup solution in case of trouble.

The suggested way to implement this is to have a setup question to fix the clock rate.
It should proceed as follows:

"Graphic clock" ? with choice "Fail safe" or "Optimal"

When Fail safe is selected the GCLK will be programmed to 45MHz a safe frequency which is set in the one program by InitChip_SP (inittable reg 40h 41h). When Optimal is selected GCLK will be programmed according to DRAM bank 0 type. Here is the value we have found as maximal to be OK with most of the DRAM. For FPM never more than 50/55MHz for EDO-70 75MHz and finally for EDO-60 85MHz. To set optimal GClock the ProgGclkAuto proc can directly be used since it get dram type from dram type register (assumed to be already programmed).

See AnyBootInitTable index 40h,41h, ProgGclkAuto

Note: All the video memory controller (type and timing) is setup in the VGA BIOS and does not require any system BIOS support.

For testing correct programming of the graphic clock the dclk.exe dos program is provided (run dclk 255).

8.EVMINI and GLORIA Boards considerations

With the evaluations boards many features are controlled through the STPC GPIO register (index 0x7b). This GPIO internal register is externally latched and at power on the internal readback value doesn't match the external one, so it must be programmed as soon as possible. The value is 0xFE for reason that will be discuss below.

See AnyBootInitTable index 0x7b

8.1.ATX-Off

The ATX off is mapped to gpio bit 0. Writing a one to these bit will simply switch off the power supply. That's why the init value bit 0 is 0. Note that this bit must cleared prior to start the OnChip VGA BIOS or in any case prior code that use the GPIO.

APM aware BIOS should implement the ATX off for switching off all device's APM call.

8.2.TV output related

Bit 1 Is STV011x reset line.
Bit 2 Is STV011x i2c SDA line.
Bit 3 Is STV011x i2c SCL line.
Bit 4 Is Internal/external (1/0) DCLK control.

Those lines are under control of the VGA BIOS and just require to be initialized to "1".

8.3.Flash 12V enable

GPIO bit 7 when written to 1 enable the 12V to the flash part.

9. INT A1h Link with VGA BIOS & OS Drivers

9.1. Purposes and goals

The goal of this software interrupt service that must be implemented by the system BIOS is to provide a way to the VGA BIOS and OS driver's to know and modify the status of Chip specific features controlled through the system BIOS CMOS setup.

It concerns Frame buffer size, TV output type, TV connector type and DRAM Bank 0 type & speed.

The frame buffer size is used by OS drivers to know if the amount of graphic memory is large enough to setup a given mode or not. If it is not, the driver can then update it in the CMOS RAM and force the system to reboot to take the change into account.

The TV output related parameters are used by the VGA BIOS to program the VGA controller and TV encoder to operate in the appropriate TV mode and to match also the selected TV connector. It may also be used by OS driver to enable special functions or settings only available when using TV.

The DRAM type and speed is only used by the VGA BIOS to program the graphic DRAM controller type & timings (notice this parameter will never require to be updated in CMOS setup).

The code in inta1.inc and inta1.def files is given as a basic starting point and as an example. But by the time the interrupt call does what it requires to do there is no restriction on the way to implement it.

To test the inta1 implementation the testa1.exe W95/DOS program is provided.